NASA Contractor Report 182049
ICASE Report No. 90–38

ICASE

A NETWORK FLOW MODEL FOR LOAD BALANCING IN CIRCUIT-SWITCHED MULTICOMPUTERS

Shahid H. Bokhari

Contract No. NAS1–18605
May 1990

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665–5225

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

A Network Flow model for Load Balancing in Circuit-switched Multicomputers*

Shahid H. Bokhari

Department of Electrical Engineering

University of Engineering & Technology, Lahore, Pakistan

and

ICASE, NASA Langley Research Center

Hampton, Virginia

Abstract

In multicomputers that utilize circuit switching or wormhole routing, communication overhead depends largely on link contention–the variation due to distance between nodes is negligible. This has a major impact on the load balancing problem. In this case there are some nodes with excess load (sources) and others with deficit load (sinks) and it is required to find a matching of sources to sinks that avoids contention. The problem is made complex by the hardwired routing on currently available machines: the user can control only which nodes communicate but not how the messages are routed.

Network flow models of message flow in the mesh and the hypercube have been developed to solve this problem. The crucial property of these models is the correspondence between minimum cost flows and correctly routed messages. To solve a given load balancing problem, a minimum cost flow algorithm is applied to the network. This permits us to determine efficiently a maximum contention free matching of sources to sinks which, in turn, tells us how much of the given imbalance can be eliminated without contention.

*Research supported by the National Aeronautics and Space Administration under NASA contract NAS1-18605 while the author was in residence at the Institute for Computer Applications in Science & Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225.

1 Introduction

The recent introduction of circuit-switched communications in multiple computer systems has greatly reduced communication overhead but, at the same time, has created new problems as far as efficient execution of programs is concerned. This is because the overhead of communication in these machines depends very slightly on the distance (i.e. number of communication links) between two nodes and very heavily on link contention (i.e. pairs of source-sink paths sharing an edge). Prior research on minimizing performance degradation due to interprocessor communications has not considered this contention issue. Users of circuit-switched machines are thus faced with many new and challenging problems.

In this paper we will address the problem of load balancing in circuit-switched machines. We will show that edge contention may cause heavy overhead when balancing load in a circuit-switched multicomputer system and will go on to develop an algorithm that minimizes this contention. We will solve this problem for two types of interconnection structures: the mesh, as typified by the Symult 2010 machine and the hypercube, whose examples include the Intel iPSC-2 and iPSC-860.

This paper is organized as follows. In the next section we discuss the load balancing problem in detail and state the assumptions under which we will solve the problem. Section 3 discusses the mesh interconnection scheme and the routing strategy used on this scheme. In Section 4 we discuss the hypercube interconnection and the well known 'e-cube' routing strategy. In Section 5 we will describe how these communication strategies are actually implemented in multicomputers and will show how edge contention arises in circuit-switched machines. Measured timings of overhead due to edge contention on the Intel iPSC-860 hypercube will be presented.

Our solution to the load balancing problem relies on network flow algorithms. In Section 6 we therefore briefly cover those aspects of networks that are necessary for the ensuing exposition. In Sections 7 and 8 we present our network flow model for mesh and hypercube connected machines, respectively, and show how these models may be used to solve efficiently the load balancing problem. We conclude in Section 9 with a discussion on future research directions. Appendix A outlines the minimum cost flow algorithm and Appendix B contains a proof of correctness of the hypercube flow model.

2 The Load Balancing Problem

We will assume that some distributed computation is taking place on our multicomputer system. The load on each computer varies slowly as new subcomputations are created and/or destroyed. This is typically the case in distributed simulations and in distributed search problems. Eventually some processors have excess load, others have deficit load and the remaining are neutral. At this point we would like to redistribute the load so as to better utilize our computational resources. This load balancing, which involves sending load from overloaded processors (sources) to underloaded processors (sinks), should be carried out with due regard for communication overhead so that it is accomplished as quickly as possible. Since we have assumed that our multicomputer system employs circuit-switched communications with a fixed routing algorithm, our prime concern is to minimize the number of paths between sources and sinks that share edges and thus minimize the impact of link contention.

Figure 1 describes this state of affairs. We show a multicomputer system with some unspecified interconnection and some unspecified (but fixed) routing strategy for interprocessor communications. There are 3 sources and 2 sinks. We can create 2 source-sink pairs and have several ways to proceed. The three parts of this figure show that some source-sink matchings* are better than others in that they result in no link contention. The problem is to efficiently find this contention free pattern of communications.

2.1 Problem Formulation

The problem that we set out to solve in this paper is stated as follows: What is the largest amount of imbalance that can be eliminated without contention? How may this be done: which excess processor should send to which deficit processor?

*We use the term 'matchings' in the informal sense of 'pairings'. In a strict graph-theoretic sense, these could be considered matchings in an imaginary complete graph whose nodes are the processors of the system.

We will make the following assumptions.

1. There is unit load imbalance. Each processor has either one unit of excess load or one unit of deficit, or is neutral.
2. The global state of the multiprocessor is known.
3. There is a fixed routing algorithm.
4. Communication overhead is due to link contention. Distance effects are negligible.

In Sections 7 and 8 we construct flow networks that correctly model routing of messages in meshes and hypercubes. The crucial property of these models is the correspondence between minimum cost flows and correctly routed messages. To solve a given instance of a load balancing problem, we apply a minimum cost flow algorithm to the network that represents that instance. The resulting flow then provides an answer to the load balancing question posed above.

3 Meshes

In the mesh interconnection scheme each processor is considered to be located on an integer mesh, with connections between processors that are one coordinate apart. Thus processor $< i, j >$ is connected to processors $< i, (j \pm 1) >$ and $< (i \pm 1), j >$. There may be 'wraparound' at the edges of the array. The Illiac-IV [1] is an early example of this interconnection. Current examples include the PAX [12, 13], the Symult 2010 [17] and the iWARP [4].

When sending messages from node $< x_s, y_s >$ to $< x_d, y_d >$, one possibility is to use a 'row-column' column routing strategy. According to this strategy, the message first travels along a row to the correct column and then along a column to the correct node. That is, $< x_s, y_s > \rightarrow < x_d, y_s > \rightarrow < x_d, y_d >$. This strategy is used on the Symult 2010. Other strategies are possible, e.g. 'column-row' or 'staircase'. We will assume the row-column strategy in our analysis of meshes.

The important aspect of routing on meshes is that it is outside the user's control. Thus there may be edge conflicts between the paths specified for two or more pairs of communicating nodes. The 'row-column' strategy will insist on these paths, even if many alternate paths exist. This is illustrated in Figure 2.

4 Hypercubes

A hypercube of dimension d has 2^d processors labeled 0 to $2^d - 1$, with a connection between two processors if and only if the binary representations of the labels of these processors differ in exactly one bit. Hypercubes that are commercially available include the Intel iPSC-2 and iPSC-860, the Thinking Machines CM-2, and machines made by Ametek and N-Cube.

The routing strategy commonly used in hypercubes is the so-called 'e-cube' algorithm [18]. According to this strategy, a message is always transmitted to the processor that more closely matches the binary representation of its destination (with comparison begun at the right hand side of the labels). As in the case of meshes, other strategies are possible, however this 'e-cube' strategy is used in all the commercially available machines mentioned above.

Edge contention can arise when two or more paths share an edge. This is illustrated in Figure 3, which shows four paths that share a single edge. It can be verified that all paths follow the 'e-cube' algorithm. The labeled rectangles in this figure represent processors of the hypercube. The unlabeled rectangles represent the hardware that interfaces with the communication network. This illustrates an important feature of most modern multicomputers: processing nodes are not burdened with the task of interprocessor communications. A message passing through node 0000011 will not interfere with the computation being carried out in node 0000011.

5 Interprocessor Communications

The above discussion has focussed on the mesh and hypercube interconnection structures and the routing strategies employed on them. There are many ways that any given strategy can be implemented on a machine. In the following, we will describe how the implementations can be divided into two broad classes: *store-and-forward* and *circuit switching* or *wormhole routing.* There are finer distinctions that may be made and, to some extent, there is overlap between these classes. Nevertheless this classification serves to clarify the assumptions under which our load balancing problem is solved. A clear and detailed discussion of these issues appears in [11].

5.1 Store & Forward

In the store and forward method a message is broken up into *packets*. Packets are forwarded along the path dictated by the applicable routing strategy. Each intervening node must receive a complete packet before forwarding it to the next node in the path from source to destination. The time to communicate depends heavily on the number of links (the graph theoretic *distance*) between source and destination.

5.2 Circuit Switching or Wormhole routing

For our purposes, circuit switching or wormhole routing are essentially equivalent. A dedicated circuit is set up between source and destination according to the applicable routing strategy and data is pipelined through this circuit [8]. This is a very fine grained process, with small chunks of data a few bytes in size (these are sometimes called *flits*). Because of this pipelining, the time to transmit data depends largely on the length of the message and is relatively insensitive to the number of hops between source and destination[†].

5.3 The Impact of Link Contention

The impact of link contention on communication overhead can be illustrated by implementing the communication pattern of Figure 3 on a 128 node Intel iPSC-860 hypercube, which is a circuit-switched machine. In our experiment we simultaneously communicate between nodes 0 → 127, 4 → 79, 6 → 111, and 7 → 15. According to the 'e-cube' algorithm this requirement results in the following directed paths.

```
1:0000000→0000001→0000011→0000111→0001111→0011111→0111111→1111111
2:        0000100→0000101→0000111→0001111→1001111
3:                0000110→0000111→0001111→0101111→1101111
4:                        0000111→0001111
```

[†] There are measurable dependencies on distance that are significant for small messages [2] but negligible for our application (which involves ≈ kbyte messages).

All paths use link 7 → 15: we can do nothing to avoid this since routing is beyond our control. The impact due to this contention is shown in the plot of Figure 4 which is taken from [2]. This plot shows the time required to communicate a message according to the requirements given above, against the length of the message. In this experiment we first established path 1 alone and obtained the plot labeled '1' in Figure 4. We then established paths 1 and 2 and obtained the plot labeled '2', and so on.

It is clear from Figure 4 that edge contention has a severe impact on the overhead of interprocessor communications. When four paths share an edge the overhead is more than 100%. A more detailed set of experiments is given in [2] which also describes communication patterns for which the overhead is much greater than 100%.

Similar degradation has been observed on the older iPSC-2 hypercube [3] and the Symult 2010 mesh connected machine [7].

While the sharing of edges between two source-destination paths has a serious impact on communication overhead, it is interesting to note that shared nodes have no impact. That is, there is no overhead if the paths between two or more source-sink pairs have a common node (but not a common edge). We have been unable to measure any degradation caused by the sharing of a node between 4 paths on the Intel iPSC-2 or on the iPSC-860.

6 Network Flows

In the remainder of this paper we will show how to solve the load balancing problem by transforming it into a network flow problem. In the present section we outline those aspects of network flow theory that are essential to the discussion that follows. Detailed exposition of this material may be found in [14, 15, 19].

We will employ graphs with *capacities* and *unit costs* on their edges. The capacity on an edge is the maximum permissible amount of *flow* through that edge. The cost of a sending a flow through an edge is the amount of the flow multiplied by the unit cost. We will be concerned only with integer flows, capacities and costs. The nodes in our graphs may be designated as *sources*, *sinks*, or neither. Each source has a positive integer called *source* *capacity* associated with it that represents the maximum amount of flow that it can generate. Similarly a sink is labeled with a negative integer called *sink capacity* that indicates the maximum amount of flow that it can consume.

A flow in this graph must satisfy the following requirements.

1. no edge carries more than its capacity,
2. the amount of flow leaving (entering) a source (sink) node is less than or equal to its source (sink) capacity, and
3. for nodes that are not sources or sinks, the amount of flow leaving a node equals the amount of flow entering a node.

The *magnitude of a flow* is the sum of all the flows leaving the source nodes (which is equivalent to the sum of all flows entering the sink nodes). The *cost of a flow* in a given graph is the sum, over all edges, of the flow in an edge multiplied by the cost per unit flow of that edge.

The problem of finding a maximum flow in this graph without regard for cost is called the *Transshipment Problem.* This problem can be transformed very easily into the classical *Maximum Flow Problem* and solved in no worse than $O(N^3)$ time, for a graph with N nodes. The transshipment problem appears, at first sight, to be similar to our load balancing problem in that both have sources, sinks and neutral nodes, and require a maximum transshipment from sources to sinks. However our load balancing problem is far more complex in that edge contention as well as a prescribed routing algorithm must be taken into account.

The problem of finding a flow of magnitude $\mathcal{F}$ in this graph that has minimum cost is called *Minimum Cost Flow Problem.* This problem is solvable, for a graph with N nodes and E edges, in time $O(\mathcal{F}E \log N)$ using the algorithm of Busacker & Gowen [5] as presented by Lawler [15]. This is not a truly polynomial algorithm, since its complexity involves the magnitude $\mathcal{F}$ of the flow. When applied to our load balancing problem, however, the flow magnitudes are polynomial functions of the size of the graph, permitting us to obtain a polynomial solution to our problem. The algorithm is discussed in greater detail in Appendix A to this paper. One interesting aspect of this algorithm is that, in finding the minimum cost flow of value $\mathcal{F}$, it also reports the minimum costs flows for all integers $1, 2, \cdots, \mathcal{F}$. This is of great value to us, as will become apparent in the following sections.

7 Network Flow model for Meshes

We are now in a position to present our flow model for routing in a mesh connected multicomputer system. Once the model has been constructed and source/sink information entered into it, the minimum cost network flow algorithm can be used to find an answer to the load balancing question stated in Section 2.1.

7.1 The model

The flow model for load balancing on a mesh with 'row-column' routing is shown in Figure 5. Each diamond in this figure represents a processor. Solid edges have capacity 1, cost 0. Dashed edges have capacity 1, cost 1. There are no connections where solid edges cross. An overloaded processor is represented by a capacity 1 source with connections to the two horizontal nodes of the corresponding diamond. Similarly, an underloaded processor is represented by a capacity 1 sink with connections to the vertical nodes of the corresponding diamond. Neutral processors have no additional nodes. All edges in this network are undirected, except for the edges leading out of (into) the sources (sinks).

The construction of this network is such that

1. a correctly routed ('row-column') flow *must* have cost exactly 1,
2. an incorrectly routed flow *must* have cost > 1, and
3. flows cannot share edges.

In Figure 5, the dashed flow is correctly routed and incurs a cost of 1 unit. The dotted flow is incorrectly routed (does not follow the 'row-column' rule) and incurs a cost > 1. It is clear that if the dotted flow were to travel first along a row and then down a column it would incur exactly one unit of cost.

7.2 Bidirectional message transmission

The network flow model proposed above has undirected edges, except for those leading out of (into) sources(sinks). This model, therefore, cannot correctly represent bidirectional flows, which correspond to two messages traveling in opposite directions through the same link[‡]. However, this creates no problems for us as far as load balancing is concerned. This is because two flows traveling in opposite directions through a link can always be replaced by a zero flow, as shown in Figure 6. The solid paths in this figure show two correctly routed messages from nodes 8 to 2, and 11 to 5, respectively. These messages pass through the edge 9-10 in opposite directions. Our network model is incapable of representing this situation, since its edges are undirected and can carry only one unit of flow in one direction at a time. However, since we are only interested in load balancing, it does not matter to us if the message from node 8 ends up in node 5 and the message from node 11 in node 2. This is shown by the dashed paths in Figure 6.

We therefore conclude that for every matching of over/under-loaded processors that results in a bidirectional message transmission through a link in the multicomputer system, there exists another matching that has no bidirectional transmission *and* uses a subset of the links of the original matching. Our network flow model can be correctly applied to the load balancing problem, despite its inability to represent bidirectional flows.

7.3 Solution

It follows that if it is possible to move $\mathcal{F}$ units of load from $\mathcal{F}$ sources to $\mathcal{F}$ sinks in a mesh connected computer system, there must exist a flow of magnitude $\mathcal{F}$ and cost exactly $\mathcal{F}$ in the corresponding flow network model, and vice-versa.

To answer the load balancing question, we apply the minimum cost flow algorithm to the network and monitor the costs of flows of magnitude $1, 2, \cdots, \mathcal{F}$ that the algorithm reports. If the algorithm is able to deliver a flow of magnitude $\mathcal{F}$ with cost $\mathcal{F}$, we can rest assured that all the imbalance can be eliminated without edge contention. If the algorithm is unable to deliver a flow of magnitude $\mathcal{F}$, or delivers a flow of magnitude $\mathcal{F}$, whose cost is $> \mathcal{F}$ we simply run through our list of reported flow costs and stop at the largest flow whose magnitude and cost are equal. This is the largest amount of imbalance that can be eliminated without edge contention. In any case,

[‡] Whether this is permissible in a real machine depends on the specific hardware in question. Some machines do not permit this, others, such as the Intel hypercubes, do.

the matching of sources to sinks that the algorithm returns is precisely the matching of over/underloaded processors required to obtain a contention-free load balancing.

7.4 An example

For the configuration of sources and sinks shown in Figure 7, the flow algorithm will return the following $< flow, cost >$ pairs:$< 1,1 >$, $< 2,2 >$, $< 3,3 >$, $< 4,6 >$,$\cdots$. Only 3 units of load can be balanced without contention, since the fourth unit costs more than 1. The legal flows are shown by dashed paths in the figure. The fourth, incorrectly routed, flow is shown by a dotted path. This example also demonstrates that, although paths exist for transmitting more than three messages without contention, the 'row-column' strategy does not permit us to use them.

7.5 Complexity

The flow network corresponding to an n processor system has $N = O(n)$ nodes and $N = O(n)$ edges. The maximum flow $\mathcal{F}$ in such a network cannot exceed $n/2$. The time to run the minimum cost flow algorithm is thus $O(\mathcal{F}E \log N) = O(n^2 \log n)$. In Appendix A to this paper we show that this can be brought down to $O(n^2)$ by exploiting problem structure.

8 Network Flow model for Hypercubes

The flow network corresponding to a hypercube with 'e-cube' routing is understandably more complex than the mesh model. In Figure 8 we show the network corresponding to a dimension 3 hypercube with two sources and two sinks. The discussion that follows uses this network as an example.

8.1 The model

The flow network for hypercubes has the following properties.

1. Each processor in a d-dimensional hypercube is represented by a d-clique in the model. In Figure 8 we have a 3-dimensional hypercube and hence 8 3-cliques[§].

2. All edges in the model have capacity 1.

3. The non-clique edges have cost 0.

4. The clique edges have cost equal to the distance between differing bits in the binary labels of the two processors that they connect. For example, in Figure 8, the clique edge connecting processors 100 and 001 (and 'bypassing' processor 101) has weight 2 because 100 and 001 differ in positions 0 and 2. Similarly, the clique edge connecting processors 000 and 110 (bypassing 010) has weight 1 since 000 and 110 differ in bit positions 1 and 2.

5. Overloaded processors are represented by sources of capacity 1. There are d edges leading out of each source, one to each node of the corresponding clique. The cost of the edge directed from node X towards node $Y = i$, the index of the bit that differs in the binary labels of X and Y $(0 \leq i < d)$. For example, the edges leading from the source at node 101 towards nodes 100, 111 and 001, have costs 0, 1 and 2 respectively.

6. Underloaded processors are represented by sinks of capacity 1. There are d edges leading into each sink, one from each corner of the corresponding clique. The cost of the edge directed towards node X from node $Y = d - i - 1$, ($d =$ cube dimension).

[§] At first sight, the model in Figure 8 may seem reminiscent of Preparata and Vuillemin's *cube-connected-cycles (CCC)* interconnection [16]. This is only because 3-cubes and 3-cliques are isomorphic–there is no other similarity between the two. Our network could be called *cube-connected-cliques*, but we prefer not to use this awkward phrase.

8.2 Solution

The construction of flow model for hypercubes is such that, in a network of dimension d:

1. a correctly routed ('e-cube') flow *must* have cost exactly $d-1$,
2. an incorrectly routed flow must have cost $> d-1$, and
3. flows cannot share edges.

In Figure 8, for example, the dashed path has cost 2 and represents a correctly routed flow. The dotted path is incorrectly routed and has cost > 2.

To find out how much of a load balance can be removed without contention, the minimum cost flow algorithm is applied to this network. The maximum flow $\mathcal{F}$ for which the cost is $\mathcal{F}(d-1)$ is the answer to the question. The source-sink matching determined by this flow is the matching of overloaded to under loaded processors required to achieve a contention-free load balancing.

While this model cannot represent bidirectional flows, an argument very similar to the one given in Section 7.2 for meshes can be used to show that bidirectional flows are not necessary for the load balancing problem. Unlike the mesh model of Section 7, the correctness of the hypercube model is not obvious. We have therefore included a proof of correctness in Appendix B.

8.3 Complexity of solution

The flow network for an n node hypercube has $N = O(n \log n)$ nodes, since each processor is represented by a d-clique, and $d = \log n$. It has $E = O(n \log^2 n)$ edges, since each of the $O(n \log n)$ nodes has $O(\log n)$ edges incident on it. As in the case of the mesh, the flow $\mathcal{F}$ cannot exceed $n/2$. The time required by the minimum cost algorithm time is thus $O(\mathcal{F}E \log N) = O(n^2 \log^3 n)$. In Appendix A it is shown that this can be brought down to $O(n^2 \log^2 n)$ by exploiting problem structure.

9 Conclusions

We have solved the problem of load balancing without edge contention by transforming the problem into a network flow problem and solving it with a minimum cost flow algorithm. Although this flow algorithm is not truly polynomial for arbitrary graphs, we have shown that the flow graphs that we create are such that the load balancing problem can be solved in $O(n^2)$ and $O(n^2 \log^2 n)$ time for meshes and hypercubes, respectively.

Our approach in this paper has been to avoid contention altogether by load balancing with a series of strictly non-contending communication steps. An extension to this research would be to permit a limited amount of contention, whenever this is advantageous. This requires a careful integration of the actual overhead of contention (as shown in Figure 4) into the solution.

Among other problems that present themselves are:

1. Exploit the special structure of network flow models to further improve the time required to solve the problem,
2. Solve the problem for non-unit load imbalances, and
3. Extend to other types of networks.

Acknowledgements

I wish to acknowledge many useful discussions with Piyush Mehrotra, David Nicol, John van Rosendale, and Robert Voigt. Comments by Prasanna Kumar and C. S. Raghavendra have also been helpful. I am grateful to David Bailey for arranging access to the 128 node iPSC-860 hypercube at NASA Ames Research Center. Tom Crockett at ICASE and Leigh Ann Turner at Ames gave me generous assistance in the use of their respective hypercubes.

Appendix A
Algorithm for Minimum Cost Flow Problem

The minimum cost flow algorithm is outlined in this Appendix. While the details of this algorithm are not necessary for understanding the solution to the load balancing problem presented above, they do have a bearing on the complexity of the solution.

The original algorithm is due to Busacker and Gowen [5] and is also described in the book by Busacker & Saaty [6]. Lawler [15] presents a much improved version that uses the relabeling technique developed by Edmonds & Karp [10], and the shortest path algorithm of Dijkstra [9]. We present Lawler's version but assume that faster variants of Dijkstra's algorithm, as described by Tarjan [19] are being utilized.

1. Connect all sources (sinks) to a super-source(sink) through edges that have zero cost and capacities equal to the capacity of the respective source(sink).

2. Assume an initial flow of zero.

3. Create an auxiliary network with respect to the present flow that gives the cost of augmenting one additional unit of flow from the super-source to the super-sink. This network may have negative weight edges.

4. Apply Edmonds & Karp's relabeling to make all edge weights positive and thus permit use of Dijkstra's algorithm.

5. Apply Dijkstra's shortest path algorithm to auxiliary network to find the minimum cost augmenting path.

6. If no path exists, stop. Last flow found is maximum and has minimum cost.

7. Increment flow by 1 unit.

8. If flow has required value $\mathcal{F}$, stop.

9. Go to step 3.

This algorithm actually yields all mincost flows for flow values $1, 2, \cdots, \mathcal{F}$. The time of the main loop is dominated by Dijkstra's algorithm which is $O(E \log N)$ for a graph with N nodes and E edges. The overall complexity is thus $O(\mathcal{F} E \log N)$. Tarjan [19] describes how Dijsktra's algorithm can be modified to work in $O(E + D)$ time for graphs in which the edge lengths are small integers and the maximum distance from the source to any vertex is D.

For the mesh problem of Section 7, the auxiliary network of step 3 above will have maximum distance $D = O(n)$. As discussed in Section 7, $\mathcal{F}$ and N are also $O(n)$ so that the complexity of the algorithm for the mesh problem can be brought down to $O(n^2)$. A similar analysis for the hypercube network of Section 8 yields a time of $O(n^2 \log^2 n)$.

Appendix B
Proof of Correctness of Hypercube flow model

The three properties that the flow model for a hypercube of dimension d must satisfy are:

1. a correctly routed ('e-cube') flow *must* have cost exactly $d - 1$,
2. an incorrectly routed flow must have cost $> d - 1$, and
3. flows cannot share edges.

Our proof proceeds by induction. These properties can easily be verified for hypercubes of dimension 3 (an example appears in Figure 8). Let us assume that these properties hold for all hypercubes of dimension $\leq k$. We can construct a hypercube of dimension $k + 1$ by juxtaposing two hypercubes of dimension k and adding edges between nodes whose labels differ in exactly one bit. Without loss of generality, we assume that the two constituent cubes differ in the leftmost bits of their labels. The flow model for a hypercube of dimension $k+1$ can similarly be constructed by juxtaposing two flow models for hypercubes of dimension k. However we now have to transform each k-clique into a $(k+1)$-clique and then add edges between corresponding $k+1$st nodes. We will also have to add a $k + 1$th edge from(to) every source(sink) to(from) the new node of its corresponding clique. This is illustrated in Figure 9.

Costs per unit flow will have to be be modified, so as to obey the rules given in Section 8.1. The following observations may be made regarding these modifications.

(a) The costs on the clique edges that fall within the two original k-cubes will not change, since the definition of these costs is independent of the dimension of the cube (see Section 8.1).

(b) The new intra-clique edges will have costs $1 \cdots k$.

(c) Costs on edges leading out of sources to nodes $1 \cdots k$ of their corresponding cliques will remain unchanged. The new edge from each source to the $k+1$st node of its clique will have cost $k+1$.

(d) Costs on edges leading into sinks from nodes $1 \cdots k$ of their corresponding cliques will be increased by 1. The new edge to each sink from the $k+1$st node of its clique will have weight 0.

An important property of the 'e-cube' algorithm is that the path between two nodes always remains within the smallest subcube containing these nodes. This *convexity* property assures us that if a source and sink pair lie within one of the original two k cubes, the paths between these two will also lie wholly within that cube. A unit flow along this path will have its weight increased from k to $k+1$ because all original edges leading into sinks were increased by 1 (see observation (d), above) and no other edges wholly within the original two k-cubes were disturbed.

Now consider paths between nodes in different k-cubes. We have assumed that the two constituent cubes differ in their leftmost bits; let us identify these cubes by this bit. We thus have cube-0 and cube-1. Since the leftmost bit is the last to change according to the 'e-cube' algorithm,

- we can *never* have a path that originates in cube-0 and contains more than one node in cube-1, and

- if a path starts in cube-1 and ends in cube-0, it *must* have exactly two nodes (and hence one edge in it).

This is because once the leftmost bit has been changed no further movement is possible. As a result, we can see that all the 'e-cube' paths in our $k+1$-cube can be partitioned into the following four classes.

(i) Paths that are wholly contained in one of the constituent k-cubes.

(ii) Paths that start in cube-1 and end in cube-1. These paths must be exactly one edge in length.

(iii) Paths that start in cube-1 and end in cube-0 and are exactly one edge in length.

(iv) Paths that start in cube-0 and end in cube-1 and are more than one edge in length.

Paths of class (i) have already been shown to have cost $k+1$. It is easy to verify (see Figure 9) that paths of class (ii) and (iii) also have cost $k+1$. A path of class (iv), $N_1, \cdots, N_{p-2}, N_{p-1}, N_p$, that starts in node N_1 in the 0-cube and ends in node N_p in the 1-cube must have exactly one node (N_p) in the 1-cube. (see Figure 9 (c)). Now consider a path from N_1 to N_{p-1}. Both these nodes are in the 0-cube. The cost of this path is thus k, as proved above. The last (directed) edge in this path will be from a clique node to a sink and will have cost equal to $(k+1)-i-1$, where i is the index of the bit that differs in the binary representations of N_{p-2} and N_{p-1}. If we now change this path so that it ends in N_p, we will have removed the sink edge into N_{p-1}, which had cost $k-i$ and instead included a clique edge whose weight is the distance between the differing bits in N_p and N_{p-2} (Section 8.1, property 4). Now, by definition, N_p and N_{p-1} differ in bit k, and N_{p-1} and N_{p-2} differ in bit i. The difference is exactly $k-i$. Thus the cost of the path is unchanged so far. The final directed sink edge that we have to go through, in the modified path, has cost $=(k+1)-i-1=0$, since $k=i$ (N_p and N_{p-1} differ in the leftmost [kth] bit).

Thus we have shown that all correctly routed flows in the $(k+1)$ cube must have cost exactly k.

Now let us consider property (2), i.e. an incorrectly routed flow must have cost $> k$. Every incorrectly routed flow that lies wholly within the original two k-cubes must have cost $> k-1$. When the two k-cubes are combined to make a $(k+1)$-cube, the cost of every path is increased by 1. Hence the property is true for all flows wholly within the two subcubes of the new $(k+1)$-cube. A flow crossing from one subcube to another will have weight $> k$ using an argument similar to that used to prove property (1) above.

Property (3) is trivially true, since all edges have capacity 1.

Thus we have shown that if properties (1), (2) and (3) above are true for cubes of dimension k, they are also true for cubes of dimension $k+1$. These properties are true for cubes of dimension 3; hence they are true for all cubes.

References

[1] G. Barnes et al. The Illiac-IV computer. *IEEE Transactions on Computers*, C-17:746–757, 1968.

[2] S. H. Bokhari. Communication overheads on the Intel iPSC-860 hypercube. ICASE Interim Report 10, May 1990.

[3] L. Bomans and D. Roose. Benchmarking the iPSC/2 hypercube multiprocessor. *Concurrency: Practice and Experience*, 1(1):3–18, September 1989.

[4] S. Borkar et al. iWARP: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing 88*, 1988.

[5] R. G. Busacker and P. J. Gowen. A procedure for determining a family of minimal-cost network flow patterns. Technical Report 15, O. R. O., 1961.

[6] R. G. Busacker and T. L. Saaty. *Finite Graphs and Networks*. McGraw-Hill, 1964.

[7] S. Chittor and R. Enbody. Performance analysis of Symult 2010's interprocessor communication network. Technical Report CPS-89-19, Michigan State University, Department of Computer Science, 1989.

[8] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(7):547–553, 1987.

[9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[10] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow algorithms. *Journal of the ACM*, 19(2):248–264, April 1972.

[11] D. C. Grunwald. *Circuit Switched Multicomputers and Heuristic Load Placement*. PhD thesis, Department of Computer Science, University of Illinois, 1989.

[12] T. Hoshino. *PAX Computer:High Speed Parallel Processing and Scientific Computing*. Addison-Wesley, 1989.

[13] T. Hoshino. An invitation to the world of PAX. *IEEE Computer*, 19:68–79, May 1986.

[14] T. C. Hu. *Combinatorial Algorithms.* Addison-Wesley, 1982.

[15] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[16] F. Preparata and J. Vuillemin. The cube-connected-cycles:a versatile network for parallel computation. *CACM*, 24(7):300–310, 1981.

[17] C. Seitz et al. The architecture and programming of the Ametek Series 2010 multicomputer. In G. Fox, editor, *Proc. 3rd. Conf. Hypercube Concurrent Architectures*, pages 33–36, 1988.

[18] H. Sullivan and T. R. Brashkov. A large scale homogeneous machine. In *Proceedings of the 4th. Symposium on Computer Architecture*, pages 105–124, 1977.

[19] R. E. Tarjan. *Data Structures and Network Algorithms.* SIAM CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.

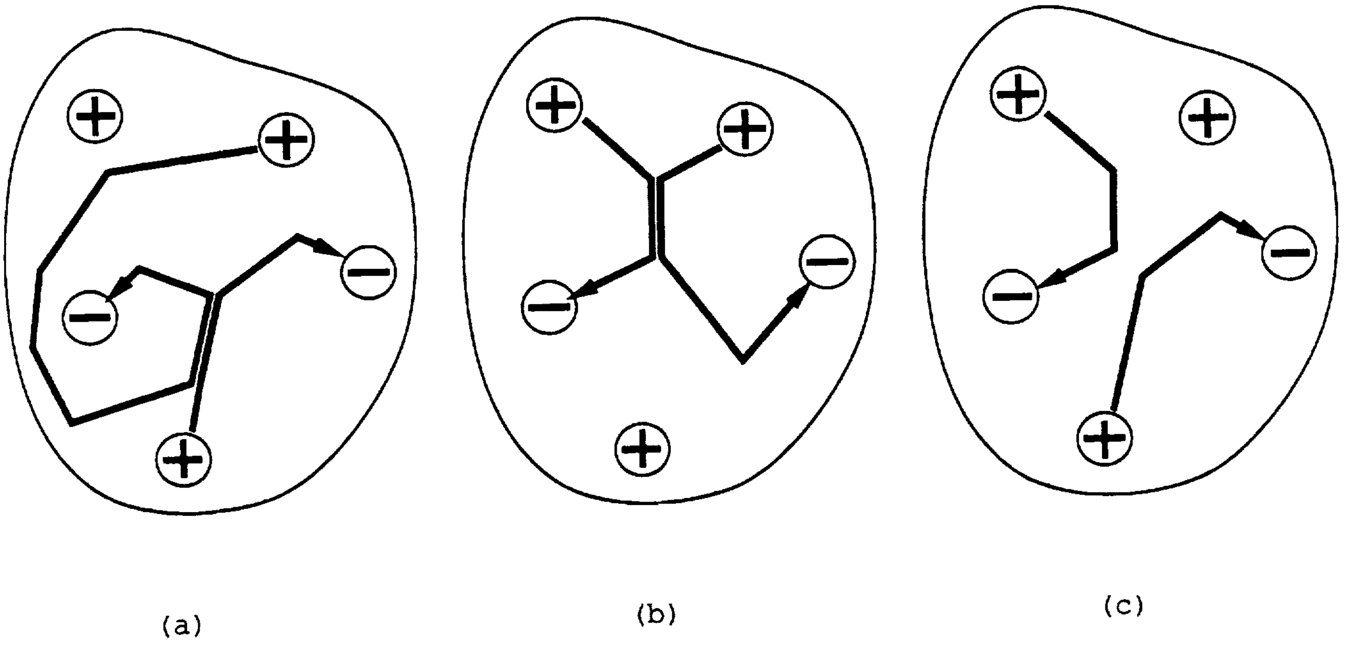

Figure 1: Given a pattern of sources and sinks, some choices of transmission are better than others. (a) and (b) incur edge contention, (c) does not.

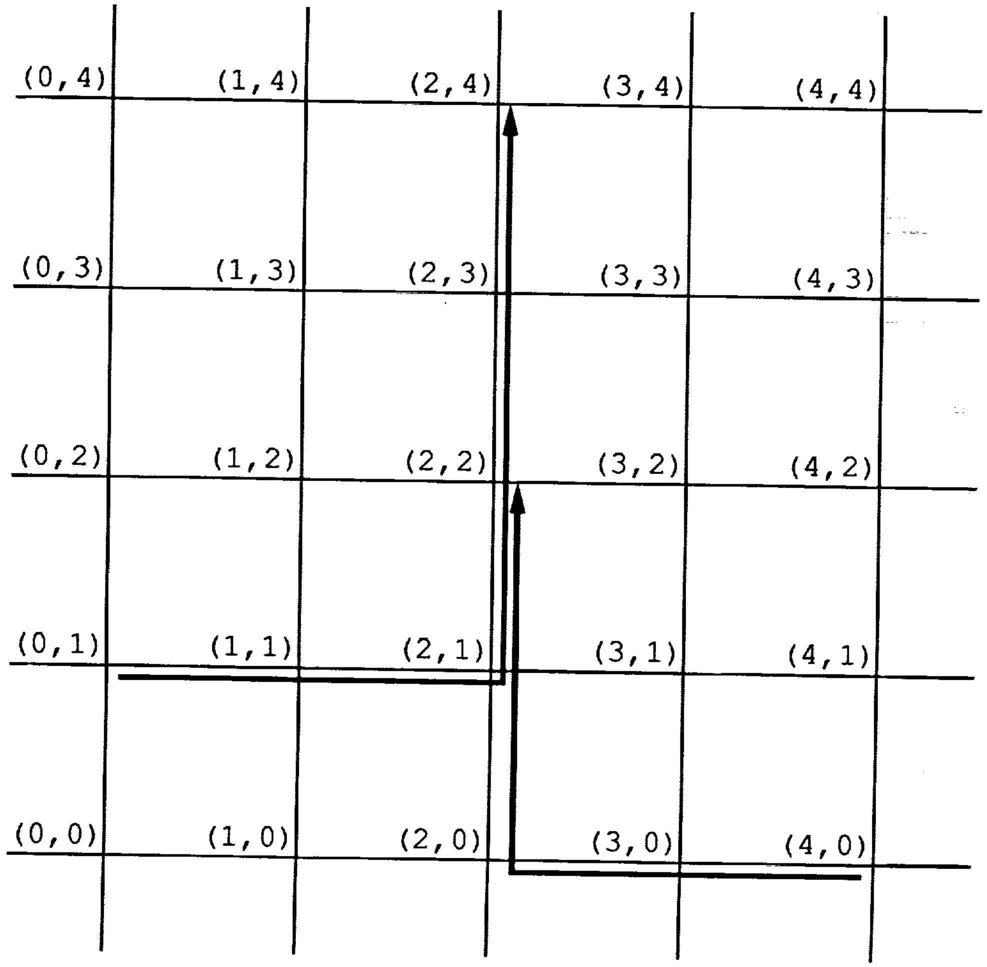

Figure 2: Edge conflicts in a mesh employing 'row-column' routing.

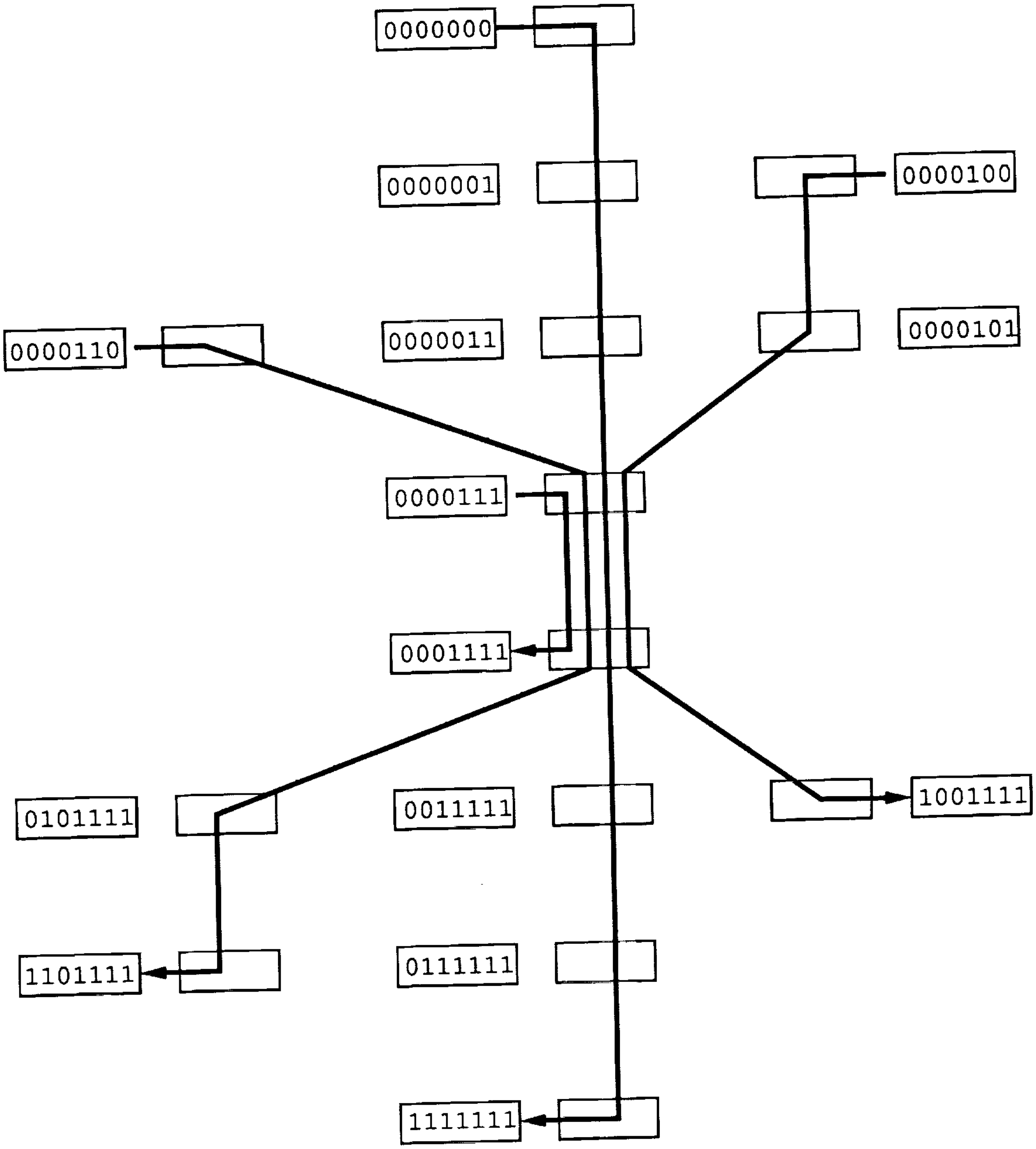

Figure 3: Edge conflict in a 128-node hypercube. All four paths are routed according to the 'e-cube' algorithm and share an edge.

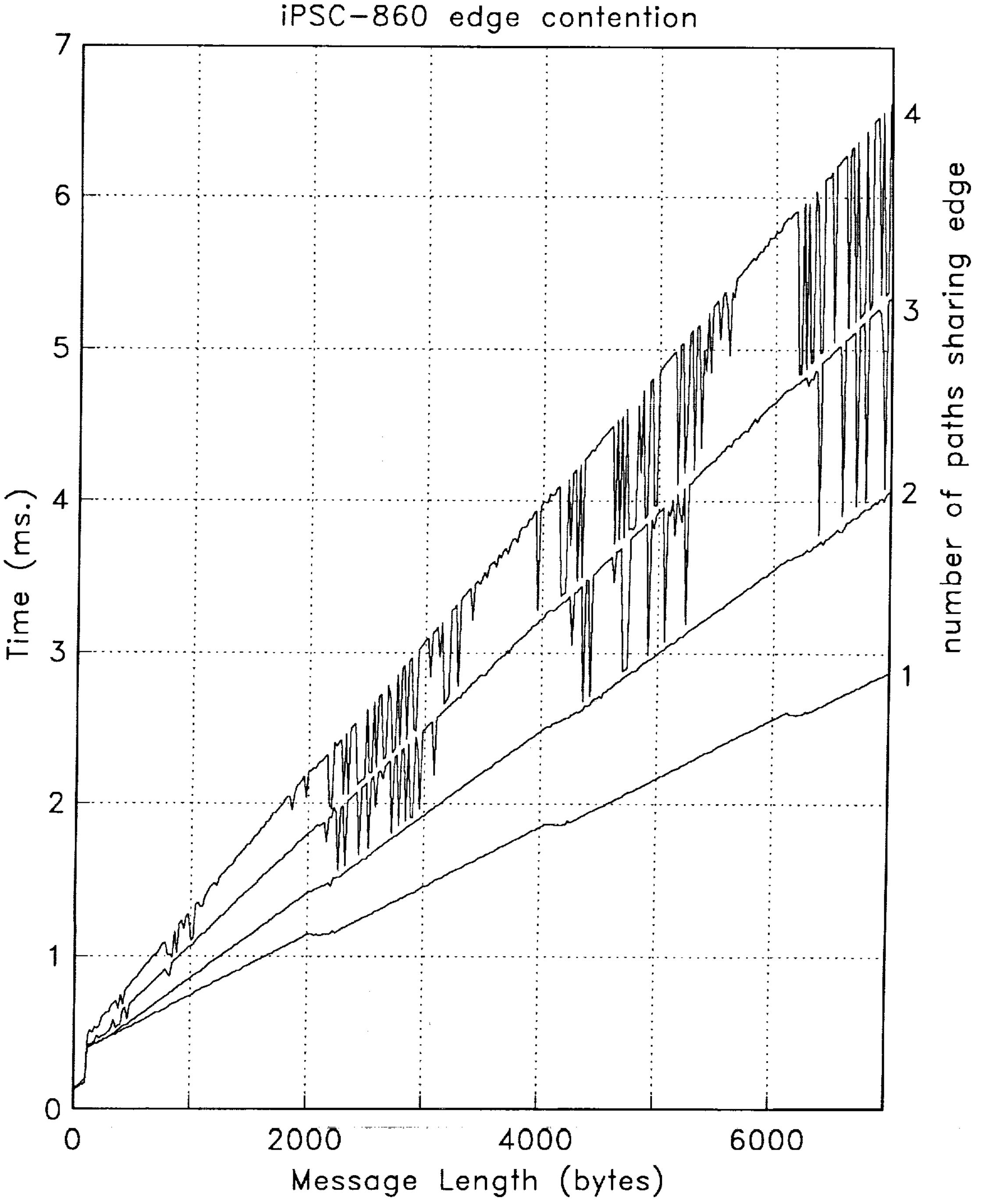

Figure 4: Impact of edge contention on communication time on the Intel iPSC-860. The paths of Figure 3 are implemented in this experiment. The label on each plot shows the number of paths that are active.

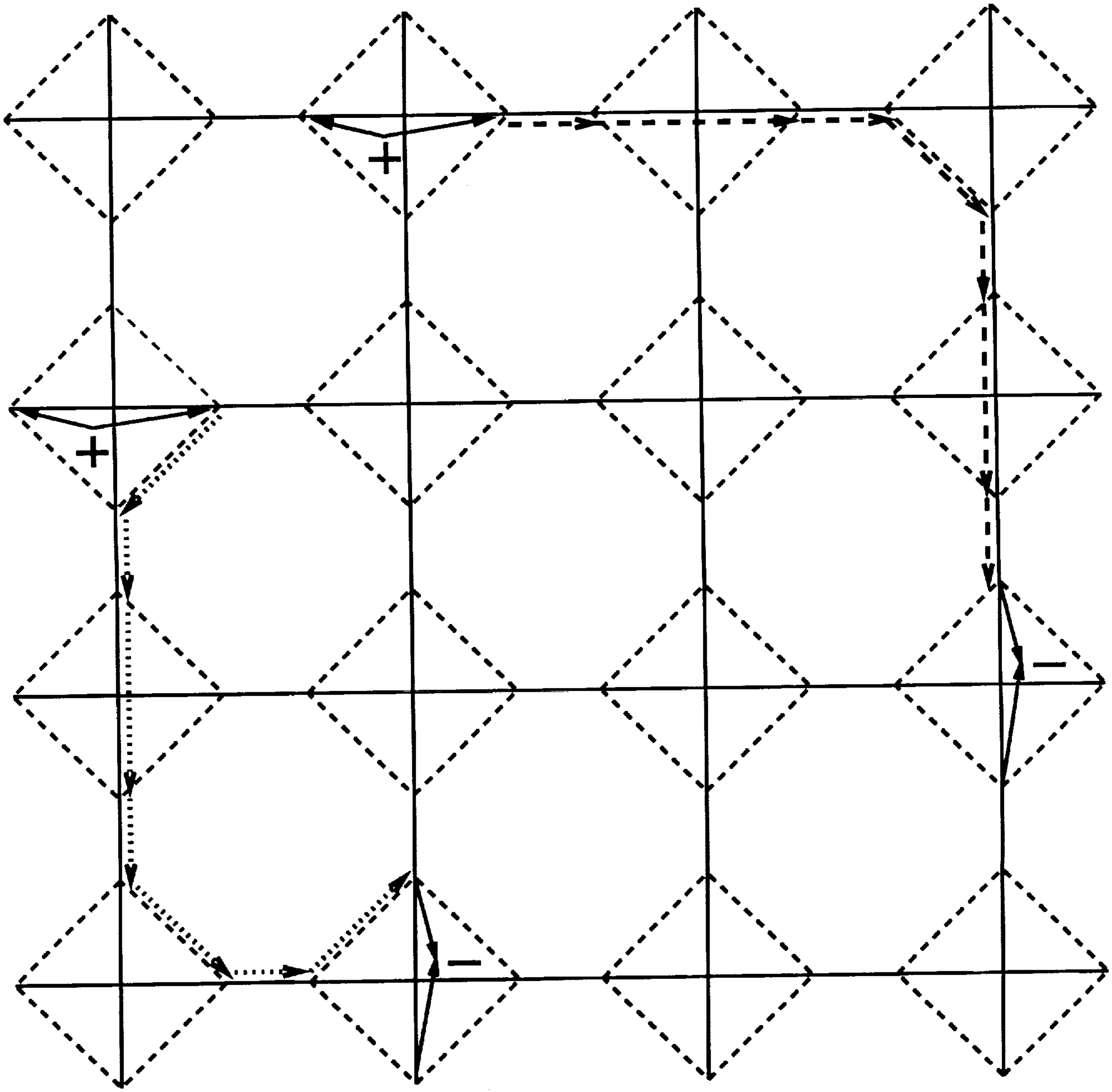

Figure 5: Flow model for load balancing in a mesh. All edges have capacity 1. Solid edges have cost 0, dashed edges have cost 1. Dashed flow is correctly routed and incurs cost = 1. Dotted flow is incorrectly routed and incurs a cost > 1.

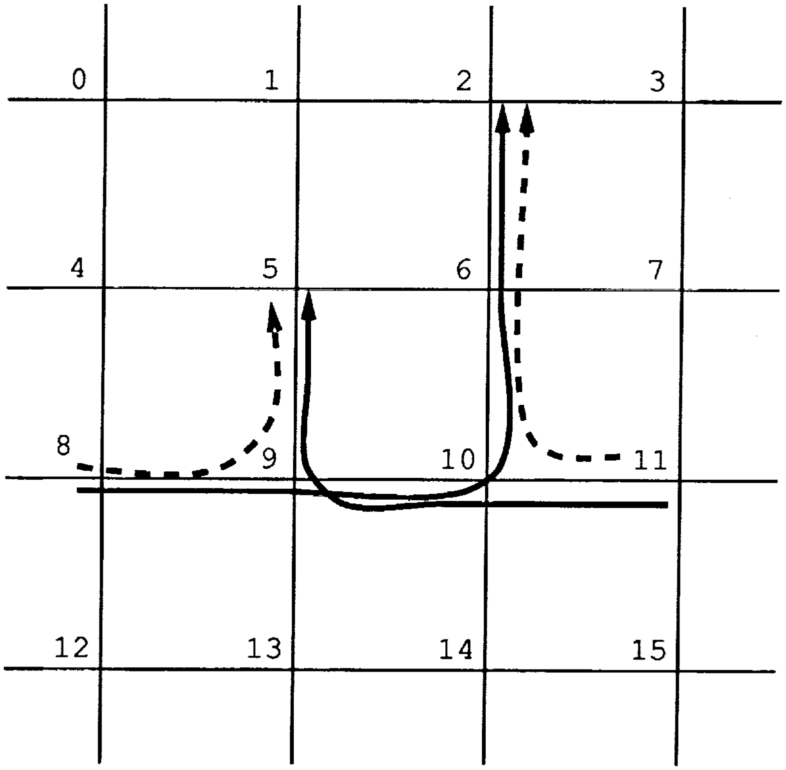

Figure 6: Any flow pattern that results in a bidirectional flow through a link can be replaced by a flow pattern that does not require bidirectional flow.

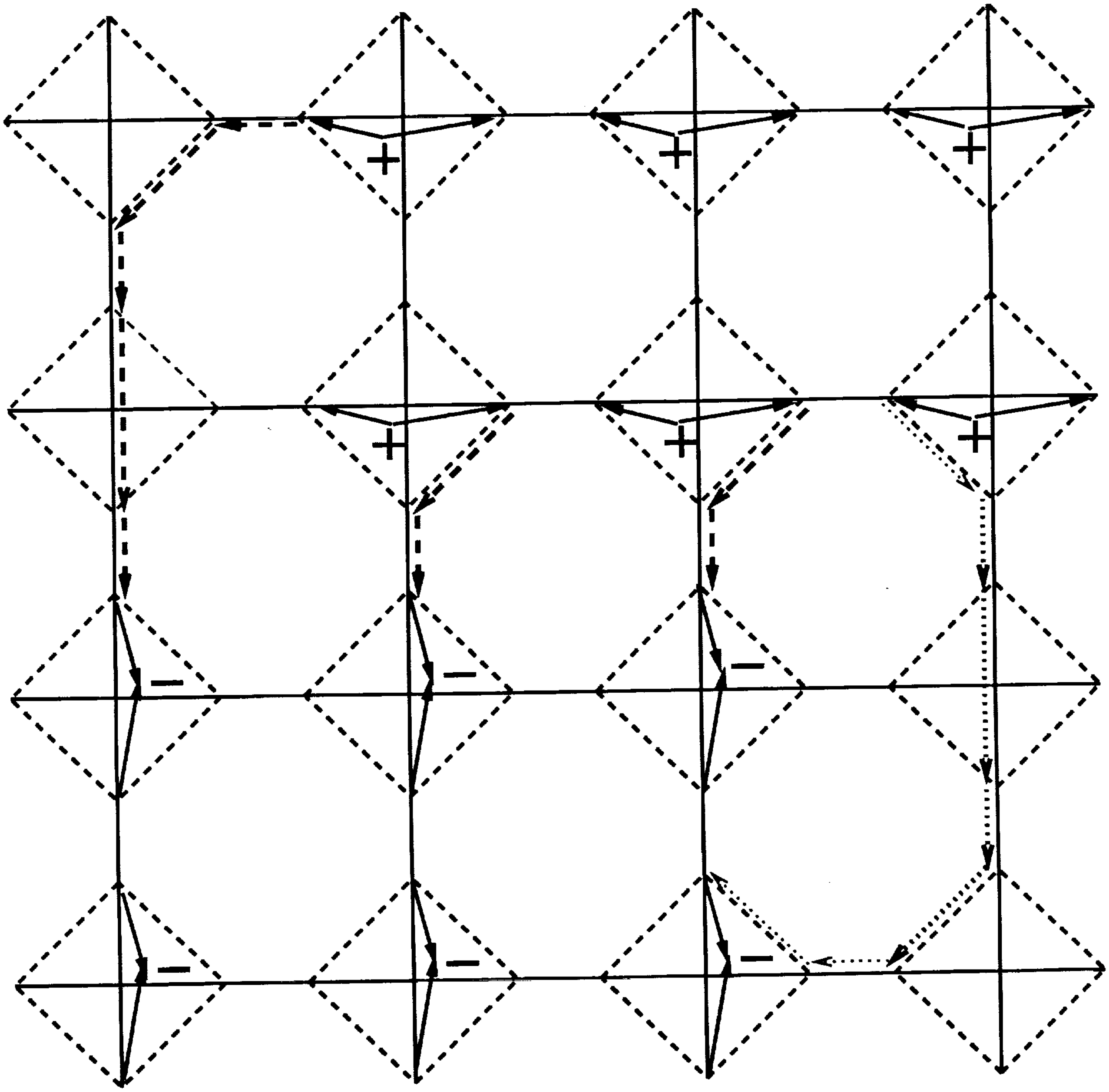

Figure 7: A mesh with 6 sources and 6 sinks. Three correctly routed flows are shown with dashed paths. The fourth flow, shown with a dotted path, is incorrectly routed. It is not possible to transmit more than 3 messages in this network without contention.

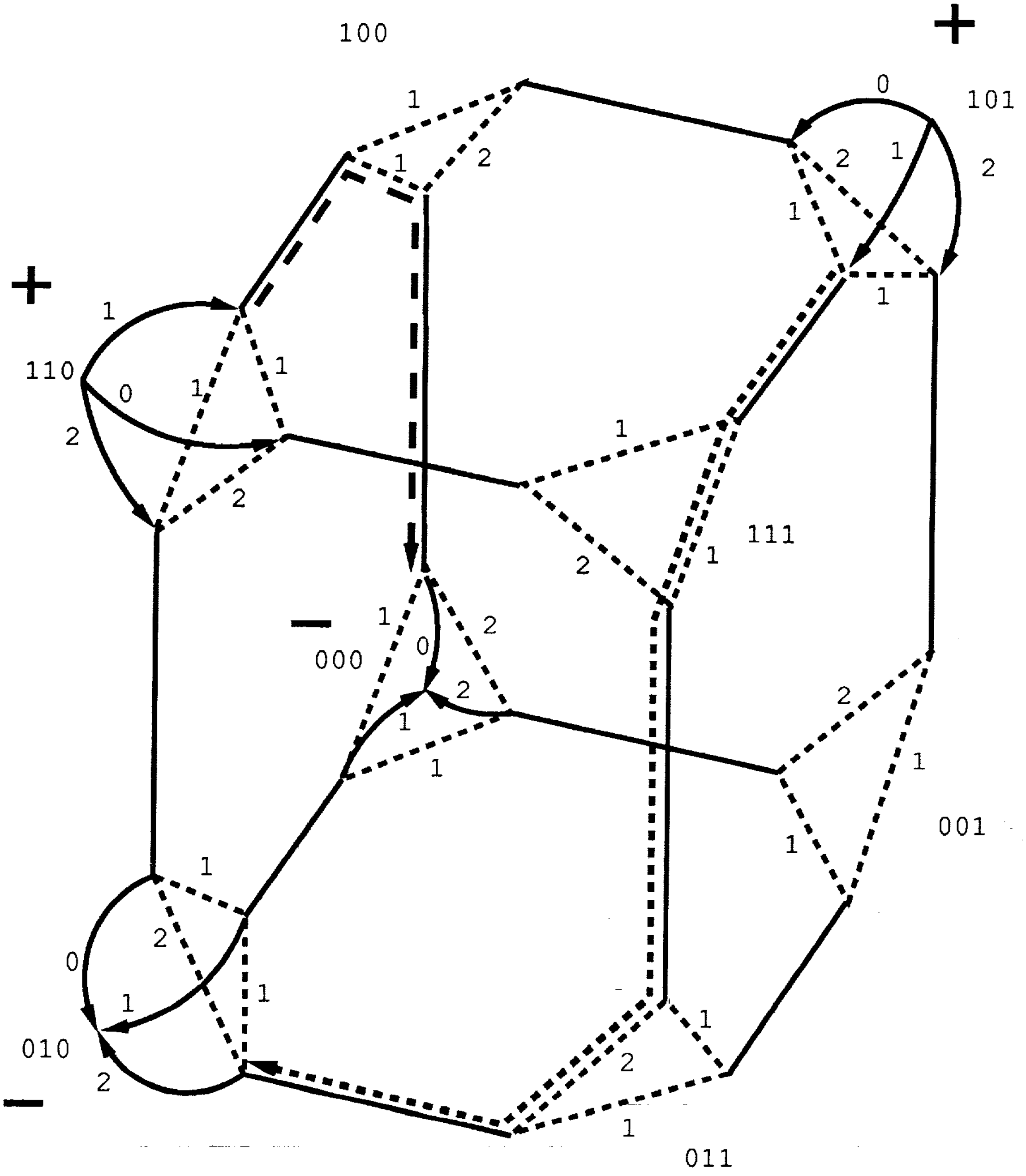

Figure 8: Flow model for hypercube of dimension 3. All edges have capacity 1, the number on each edge is its cost per unit flow. Unlabeled edges have zero cost. Dashed path indicates a correctly ('e-cube') routed flow. Dotted path indicates an incorrectly routed flow.

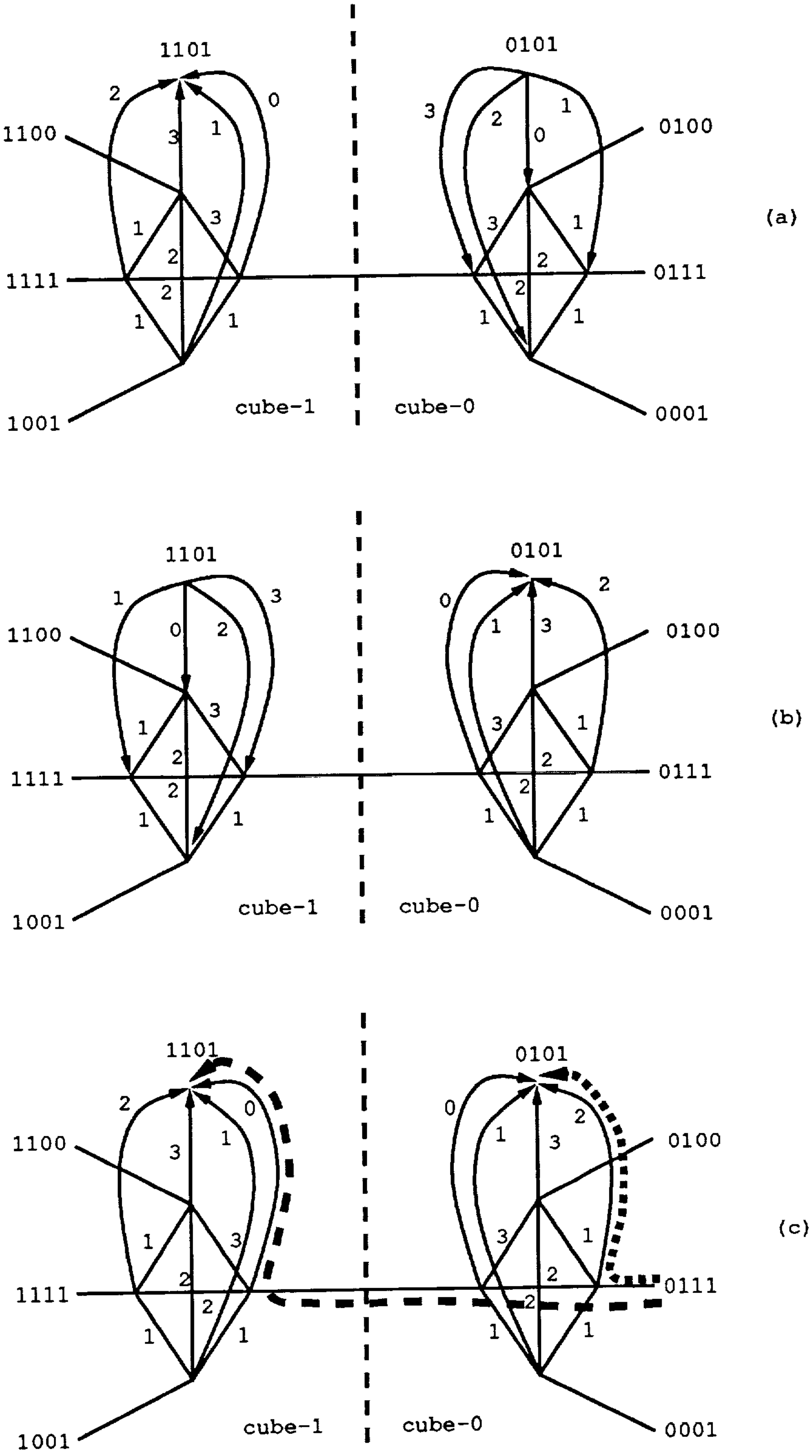

Figure 9: Examples illustrating proof of correctness of hypercube model. A 4-dimensional network is constructed by juxtaposing two 3-dimensional networks. (a) A path of length 1 from cube-0 to cube-1. (b) A path of length 1 from cube-1 to cube-0. (c) Extending a path that ends in node 0101 so that it ends in node 1101.

National Aeronautics and Space Administration

Report Documentation Page

1. Report No.
NASA CR-182049
ICASE Report No. 90-38

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle

A NETWORK FLOW MODEL FOR LOAD BALANCING
IN CIRCUIT-SWITCHED MULTICOMPUTERS

5. Report Date
May 1990

6. Performing Organization Code

7. Author(s)

Shahid H. Bokhari

8. Performing Organization Report No.
90-38

10. Work Unit No.
505-90-21-01

9. Performing Organization Name and Address
Institute for Computer Applications in Science
and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23665-5225

11. Contract or Grant No.
NAS1-18605

13. Type of Report and Period Covered
Contractor Report

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

14. Sponsoring Agency Code

15. Supplementary Notes
Langley Technical Monitor:
Richard W. Barnwell

Submitted to IEEE Transactions
on Parallel and Distributed
Systems

Final Report

16. Abstract

In multicomputers that utilize circuit switching or wormhole routing, communication overhead depends largely on link contention--the variation due to distance between nodes is negligible. This has a major impact on the load balancing problem. In this case there are some nodes with excess load (sources) and others with deficit load (sinks) and it is required to find a matching of sources to sinks that avoids contention. The problem is made complex by the hardwired routing on currently available machines: the user can control only which nodes communicate but not how the messages are routed.

Network flow models of message flow in the mesh and the hypercube have been developed to solve this problem. The crucial property of these models is the correspondence between minimum cost flows and correctly routed messages. To solve a given load balancing problem, a minimum cost flow algorithm is applied to the network. This permits us to determine efficiently a maximum contention free matching of sources to sinks which, in turn, tells us how much of the given imbalance can be eliminated without contention.

17. Key Words (Suggested by Author(s))
contention, circuit-switched machines,
hypercubes, load-balancing meshes, minimum
lost flows, network flows

18. Distribution Statement
62 - Computer Systems
66 - Systems Analysis

Unclassified - Unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 31	22. Price A03

NASA-Langley, 1990